pdpk - Uma definição de pacote de software para o ambiente Pure Data

Lucas Nascimento Oliveira [1], **Flávio Luiz Schiavoni**[1]

[1]Departamento de Computação
Universidade Federal de São João del Rei
São João Del Rei - MG - Brasil

91lucasoliveira@gmail.com, fls@ufsj.edu.br

Abstract. *This paper presents a update manager system to a Music programming environment called Pure Data. The Update management system's definition includes the definition of a package, a bundle to distribute new plugins for this environment, and a minimal repository to store plugins. In this paper we present the project decisions to create this system such as the package name's standard, package's extension, package's structure, package's descriptor and so on. Despite the fact of the present paper describes project's decision to Pure Data environment, it can be used by programmers to create a similar tool to any other software.*

Resumo. *Este artigo apresenta uma ferramenta de gestão de atualizações para um ambiente de programação musical chamado Pure Date. A definição do sistema de gerenciamento inclui também a definição de pacote, um conjunto para distribuir novos plugins, e um repositório mínimo para armazenar os plugins. Neste artigo apresentamos as decisões de projeto para criar a ferramenta, tal como o nome padrão, a extensão do pacote, a estrutura, o descritor e afins. Apesar do fato de este artigo apresentar decisões de projeto a cerca do ambiente de desenvolvimento Pure Data, ele pode ser também usado para criar um ferramenta semelhante para qualquer outro software que aceite extensões.*

1. Introdução

O desenvolvimento de software tem se mostrado uma tarefa contínua pois após o empacotamento e entrega de um sistema pode haver necessidades de manutenção no código. A manutenção no código de um sistema não é associada apenas a correção de erros pois há também a demanda por melhorias e novas funcionalidades para uma aplicação. Uma das maneiras de garantir a instalação de melhorias, manutenção do código e adição de novas funcionalidades em um sistema é prever durante o desenvolvimento a modularização de suas características. Isto permite que, caso haja uma melhoria ou adição/remoção de alguma funcionalidade que irá modificar o sistema, não seja necessário uma nova instalação pois torna-se necessário atualizar apenas o trecho de código responsável por esta funcionalidade.

O desenvolvimento de sistemas modulares é hoje uma prática comum utilizada em diversos aplicativos como: ambientes de desenvolvimento de software, como Netbeans [Myatt et al., 2008] e Eclipse [Eclipse.org, 2003]; ferramentas para produção musical, como Ardour e Pure Data [Zmölnig, 2001]; navegadores como o Firefox [Eduardo, 2013] e Chrome; ferramentas de edição de imagens como o GIMP; ferramentas de gerenciamento de conteúdo web como o Drupal; entre outras. Estas ferramentas modularizaram

seu desenvolvimento de maneira que novas funcionalidades podem ser adicionadas ao sistema e atualizações do sistema possam ser feitas por meio de plugins.

Para facilitar a atualização destes sistemas, boa parte destas ferramentas possui gestão de atualizações em formas de pacotes, permitindo a instalação ou remoção de alguma atualização de forma prática. Entre as aplicações supra-citadas, o ambiente Pure Data possui a capacidade de ser estendido por meio de plugins mas não possui até o momento um gerenciador de atualizações que permita ao usuário estender seu sistema de maneira transparente.

Diante deste cenário, o presente artigo apresenta o projeto que propõe o desenvolvimento de uma ferramenta que permitirá ao usuário do Pure Data gerenciar as extensões deste ambiente de maneira simples e prática por meio de um gerenciador de atualizações.

Desenvolver tal ferramenta implica em criar todo o suporte tecnológico necessário para a atualização automática incluindo a) a definição de um pacote de software para a distribuição do plugin, b) a criação de uma ferramenta para o empacotamento do plugin, c) a definição de um repositório local para a instalação e remoção de plugins, d) a definição de repositórios remotos para a atualização automática de plugins, e) o desenvolvimento de ferramentas para instalação, remoção e atualização do repositório local e f) o desenvolvimento de ferramentas para a manutenção do repositório remoto. Neste trabalho, discutiremos a definição de pacotes de software para um ambiente baseado em plugins. Estes componentes de um sistema de atualização automática serão melhor discutidos na Seção 3 deste documento.

Acreditamos que, apesar de este trabalho ser um estudo de caso de atualização automática para o ambiente Pure Data, as definições necessárias para a conclusão do mesmo são bastante genéricas e poderão auxiliar outros desenvolvedores a criarem ferramentas de gerenciamento de atualização para outros sistemas.

Este trabalho está organizado como segue: a Seção 2 apresenta o ambiente Pure Data e seus plugins, a Seção 3 apresenta os componentes de um Sistema de Atualização Automática de Software, a Seção 4 traz o desenvolvimento alcançado por este projeto até o momento, a Seção 5 apresenta os resultados alcançados até o momento e a Seção 6 traz a conclusão e trabalhos futuros desta pesquisa.

2. Pure Data

O Pure Data[Puckette, 2007] (Pd) é um ambiente de programação musical visual, focado em permitir que artistas, músicos e pesquisadores possam programar softwares graficamente, sem utilizar linhas de códigos. Este ambiente é utilizado para processar e gerar som e vídeos e foi projetado também para permitir a fácil colaboração em rede para músicos e colaboradores conectados via LAN ou mesmo distribuídos ao redor do mundo para criar música ou outros projetos em tempo real. Este ambiente é ainda utilizado para o ensino e aprendizado de processamento de mídia e programação visual assim como para o desenvolvimento de soluções de sistemas complexos de larga escala [Brinkmann et al., 2011] como jogos, sintetizadores ou outras aplicações que necessitam de recursos sonoros.

A programação em Pure Data é feita de maneira modular onde cada funcionalidade é fornecida por um objeto representado visualmente por uma "caixinha"[Puckette et al., 1996] (ver Figura 1). Vale lembrar aqui que a definição de um Objeto em Pure Data não segue a definição de Orientação a Objetos em linguagens de programação como C++ ou Java. Um objeto pode se conectar a outros objetos por meio de suas entradas e saídas, como um componente de software. Esta conexão é feita por

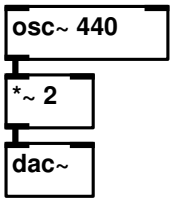


Figura 1: Exemplo de um *patch* do Pure Data onde um Oscilador de 440 Hertz tem sua saída multiplicada por 2 e então conectada ao conversor digital analógico que representa a saída de som do computador.

meio de *inlets* (entradas) e *outlets* (saídas) dos objetos. A conexão de um objeto a outro é feita visualmente por meio de uma "cordinha".

Um programa em Pure Data é chamado de *patch* (remendo) e pode conter dezenas de objetos e conexões. A Figura 1 apresenta um *patch* com 3 objetos conectados entre si. Por esta figura é possível notar que nem todos objetos possuem entradas e saídas e que nem todas as entradas e saídas dos objetos são utilizadas para comporem um *patch*.

Este ambiente foi desenvolvido usando 2 linguagens de programação distintas sendo o *engine* feito em C e sua GUI feita em Tcl/tk sendo que *engine* e GUI se conectam por um socket. Há atualmente distribuições deste ambiente para diferentes sistemas operacionais como Linux, Unix, MacOS, Android e Windows. Além de conter vários objetos em sua distribuição oficial (chamada *Vanila*), o Pd pode ser estendido por meio de *plugins*.

2.1. Plugins do Pure Data

Um plugin é um programa de computador que serve como um extensor de funcionalidades, usado para adicionar funções, prover alguma funcionalidade especial ou específica a outro programa. Uma aplicação pode utilizar tal técnica por certos motivos, um dos principais é permitir que desenvolvedores de software externos estendam as funcionalidades de um produto. Um produto que permite sua extensão por meio de plugins é chamado de host ou hospedeiro, e provê serviços que o plugin pode usar, incluindo uma forma da extensão registrar a si mesma no hospedeiro e um protocolo para a troca de informações entre eles. Os plugins dependem de tais serviços, e geralmente não podem ser executados sem um hospedeiro. O hospedeiro por sua vez é independente, de forma que é possível adicionar e atualizar plugins dinamicamente, sem a necessidade de efetuar alterações no hospedeiro em si. O Pure Data, assim como outras aplicações de áudio, é um hospedeiro para tipos específicos de plugins.

Um plugin em Pure Data pode ser feito de três maneiras[Zmölnig, 2001]: a) no próprio ambiente Pure Data como um patch, sendo comumente chamado neste caso de abstrações, b) em Tcl/tk no caso de plugins que adicionam funcionalidades a GUI, também chamados de plugins de GUI e c) em C para funcionalidades que envolvem processamento de sinais, também chamados *externals*. É ainda possível desenvolver um plugin utilizando todas as possibilidades anteriores, como, por exemplo, desenvolver um plugin que possue uma GUI em TCL/Tk, a ajuda em Pure Data e processamentos em C.

Os plugins podem ser instalados no mesmo diretório do Pure Data, em um diretório do sistema compartilhado com todos os usuários ou em um diretório local do próprio usuário. Plugins do tipo abstrações possuem uma extensão **.pd** e são independentes do sistema operacional. Plugins de GUI, por serem feitos em TCL/Tk, são independentes do Sistema operacional e possuem obrigatoriamente um nome terminado com **-plugin** e extensão **.tcl**. Plugins externos dependem do Sistema operacional onde eles foram compilados e possuem extensões diferentes para cada Sistema, como, por exemplo,

.pd_linux, **.pd_darwin** (MacOS) e **.dll** (Windows).

Um plugin pode ainda conter um arquivo de ajuda que explica seu funcionamento. Arquivos de ajuda são feitos em Pure Data e possuem o mesmo nome do plugin mas com terminação **-help.pd**. Mais sobre escrita de plugins para o Pure Data pode ser encontrado em[Zmölnig, 2001].

3. Gerenciamento de Atualização de Software

Utilizar ferramentas para o Gerenciamento de Configuração de Software durante o desenvolvimento de um sistema de software é uma prática recomendada ao desenvolvedor. O gerenciamento de configuração de software pode ser feito utilizando um conjunto de ferramentas de apoio para o desenvolvimento e manutenção do mesmo. O controle de versão, o controle de mudança e a auditoria das configurações são alguns dos atributos dessa gestão. Essa gerência tem por objetivo analisar o que mudou e quando mudou, o porque dessa mudança, o que tal mudança fez e o que podemos fazer com essa mudança. Essas questões representam as atividades praticadas pelo gerenciamento de configuração de software, tal como o controle de versão, é capaz de dizer o que mudou e quando mudou, o controle de mudanças é capaz de atribuir os motivos a cada uma das mudanças, a auditoria por sua vez responde quem fez a mudança e o que podemos fazer com ela [MENESES and Huzita, 2009].

Uma vez que houve mudança em um sistema, o próximo passo é atualizar as instalações existentes de maneira automática e transparente ao usuário. Para isto, as mudanças auditadas pelo Gerenciamento de configuração de Software são refletidas em pacotes de software que poderão ser instalados por meio de ferramentas automatizadas que se utilizam de repositórios para proceder esta atualização. Estes detalhes serão melhor discutidos nas próximas Subseções.

3.1. Pacotes

Um pacote de software é um arquivo de certo formato constituído de um conjunto completo e documentado de programas que compõem a instalação de uma nova funcionalidade ou uma atualização do sistema. O pacote contém todos os arquivos necessários, sejam eles scripts, configurações, dados, documentação e outros, para a instalação do software, além de todas as informações necessárias para a instalação em si, a sua instalação, desinstalação e configuração. O pacote vem em um formato de arquivo próprio para ser instalado por um sistema de gestão de pacotes, manualmente ou instalado autonomamente. O termo pacote aqui difere do utilizado no Diagrama de pacotes da UML onde o mesmo pode ser considerado um conjunto de classes e interfaces.

Um exemplo de utilização de pacotes de software para atualização é o das distribuições Linux, que utiliza softwares em pacotes, manipulados por um gerenciador de pacotes. Essa gestão de pacotes simplifica o processo de instalar e desinstalar aplicativos neste Sistema operacional[Ferreira, 2006].

Para refletir a auditoria de modificação feita pelo Gerenciamento de configuração de Software, muitos sistemas utilizam o nome do pacote como principal referenciamento ao mesmo. Normalmente os nomes de pacotes refletem seu órgão desenvolvedor, como o RPM da Red Hat e o DEB da Debian, ou também especificamente à sua utilidade e utilização. Por esta razão, o nome do pacote é geralmente formulado pelo nome e versão e arquitetura, permitindo identificar um pacote apenas por este atributo.

Pacotes podem ainda possuir metadados, informações essenciais ao uso como o nome completo, descrição, utilização, versão/revisão, fabricante, website, uma lista de

dependências caso necessária, arquitetura e licença. Esses metadados auxiliam à gerência do pacote tanto manual como automatizada. Os metadados do pacote compõe o descritor do pacote, que faz parte da estrutura do mesmo.

3.2. Sistema de Gerenciamento de Software

Um sistema de gerenciamento de software consiste em um conjunto de ferramentas para executar a instalação, remoção, configuração e atualização de pacotes de software. Esse sistema é chamado de gestor de pacotes, e são comumente usado em sistemas Linux. O gerenciador de pacotes obtém os pacotes de software de repositórios, resolve as dependências e os instala no sistema. O gerenciador de pacotes também facilita a remoção de pacotes ou a atualização dos mesmos. O número de pacotes disponíveis para instalação depende de quais repositórios existem adicionados aos sistema[Ferreira, 2006][Novell, 2011].

3.3. Repositórios

Um repositório de software é um local, geralmente um ou mais servidores, onde ficam armazenados pacotes de software para determinada aplicação. Estes servidores permitem que os pacotes possam ser recuperados e instalados na aplicação de maneira automática. A utilização de um servidor como repositório de pacote é um recurso muito utilizado por editores de software e outras organizações, mantendo um servidor na internet disponível para conexões HTTP ou FTP para este fim. Repositório normalmente fornecem também um sistema de indexação de pacotes que permite o gerenciamento e consulta dos pacotes pertencentes ao repositório. A utilização de repositórios públicos para pacotes de atualização de software pode transformar o gerenciamento de pacotes e atualizações em uma atividade transparente ao usuário e mais gerenciável ao desenvolvedor por evitar que o desenvolvedor envie uma atualização para cada cliente que possui sua aplicação instalada e evitar que o cliente tenha que lidar diretamente com atualizações.

3.4. Dependências

Um aspecto importante do repositório de pacotes são as relações que podem existir entre pacotes. Efetivamente, os pacotes também se relaciona com arquivos de outros pacotes, como os pacotes de aplicativos precisam de um ambiente de execução para realmente executar o aplicativo. As dependências do pacote são utilizadas para expressar tais relações. Como sendo, um pacote necessita da instalação de outro pacote previamente para que o mesmo funcione corretamente. Dependências de pacote são transitivas, podendo ser por isso necessário instalar uma quantidade relativamente grande de pacotes antes de instalar o pacote desejado. Dependências em bibliotecas são comuns e praticamente todos os aplicativos individuais de um sistema operacional podem depender de um conjunto de pacotes de bibliotecas. Os pacotes e as dependências do pacote são aspectos muito importantes das distribuições Linux, pois fornecem uma maneira modular para criar e gerir um sistema operacional e seus aplicativos. É também uma maneira eficiente de manter um sistema estável e seguro. Assim, quando uma falha afeta uma biblioteca usada por um ou vários aplicativos, atualizar um pacote singular atualizará o sistema para todos aplicativos [Novell, 2011].

4. Desenvolvimento

Este artigo apresenta os avanços alcançados até o momento no projeto que tem por objetivo desenvolver o gerenciamento de atualizações de plugins para o ambiente Pure Data. O primeiro passo feito neste desenvolvimento foi definir um formato de pacote a ser utilizado no sistema. A definição de pacote inclui a definição do nome e extensão do pacote, estrutura do pacote e descritor do pacote, conforme apresentada a seguir.

4.1. Extensão

A principal referência de um pacote é o nome do seu arquivo. Apesar de a maioria dos trabalhos estudados utilizarem como pacote um arquivo compactado (como zip ou rar), vários destes trabalhos utilizam uma extensão do nome de um pacote que se refere à sua aplicação. Exemplos encontrados de extensões de nome de arquivo são: deb para pacote da distribuição Debian, RPM é o nome de pacote da Distribuição Red Hat, nbm para plugins do NetBeans[Myatt et al., 2008], xpi para plugins do Firefox[Eduardo, 2013], crx para plugins do Chrome[Chrome, 2015], entre outras. Assim, é conveniente atribuir ao pacote proposto um nome relacionado a sua distribuição em vez de utilizar a extensão relacionada ao tipo da compactação do arquivo. Por esta razão, definimos o nome do pacote proposto para PDPK (Pure Data Package) em português: Pacote do Pure Data. Desta forma definimos a principal referência ao pacote a ser trabalhado, PDPK constará em nomes de arquivos, descritores e etc. . . .

4.2. Nome

O nome do arquivo do pacote deverá conter os atributos necessários para reconhecê-lo de maneira simples. Mais do que o plugin contido no pacote, é bastante útil ao usuário e ao desenvolvedor que o nome do pacote traga outras informações. Seguindo novamente exemplos de trabalhos relacionados, foi escolhido que o nome do pacote será composto do nome do produto juntamente com a sua versão, responsável por identificar entre versões novas e antigas do mesmo plugin, e a sua arquitetura, um aspecto importante para a verificação de compatibilidade entre sistemas. Definidos os mais importantes aspectos que irão compor o nome do arquivo do pacote de maneira legível, um exemplo de como seria tal composição é:

```
name-version-supported_architecture.pdpk
```

É importante salientar que o caractere que separará os aspectos no nome do arquivo será o hífen, e que tal caractere juntamente com o ponto, responsável por identificar a extensão do arquivo, não deverá ser utilizados em nomes de plugins. De forma alternativa, os espaços contidos serão substituídos pelo caractere subscrito, novamente por fins de legibilidade. Com o objetivo de prover um nome de arquivo fácil de identificar, manusear e ler, definimos assim o nome do arquivo do pacote.

4.3. Estrutura

A estrutura de um pacote é definida pelos arquivos que compõe o pacote e como eles são distribuídos dentro do pacote. Famosos pacotes como RPM e DEB possuem uma estrutura similar, o que permite o estudo de um certo padrão quanto a estrutura de um pacote proposto. De acordo com os pacotes estudados e também com a Norma NBR 12119[Técnicas, 1998] referente à qualidade de pacotes de software, a estrutura de um pacote consiste na composição de: arquivos que formam o produto e seus dados, descrição do produto que contém todos os metadados relacionados ao produto e seu desenvolvimento, e a documentação do usuário responsável por descrever o uso, instalação e manuseio do software. É necessário que todos esse arquivos componham um único pacote. Adicionalmente, a fim de padronizar descritores e documentações, é definido que os mesmos sejam em língua inglesa.

Estes são os três elementos requeridos à fazer parte da composição do pacote proposto. Os arquivos de documentação e descrição estarão contidos em uma pasta dentro do pacote, e os arquivos e seus respectivos dados estarão na própria fonte do pacote. Pensando nessa estrutura e novamente a fim de fornecer mais facilidade de manuseio e compatibilidade, o formato do pacote será um arquivo compressado .zip, um formato

que permite uma fácil extração, permitindo até que a mesma seja feita manualmente por usuários. Um exemplo da composição do pacote pode ser vista no Código 1.

```
1    name_of_plugin-version-architecture.pdpk
2        +- name_of_plugin.pd_linux
3        +- name_of_plugin-help.pd
4        +- name_of_plugin-plugin.tcl
5        +- Documentation Folder
6            - Descriptor.txt
7            - README.txt
```

Código 1: Exemplo da composição do pacote

4.4. Definição do descritor

O descritor do pacote é um conjunto de dados consistindo de toda informação necessária ao gerenciamento do produto. Assim a partir dos descritores estudados dos pacotes RPM e Debian, foi possível definir os dados essenciais a serem usados pelo descritor proposto, são apresentados a seguir:

- Nome: Referenciará o nome do produto contido no pacote. Assim como descrito no nome do arquivo, o nome do pacote não pode conter hifens ou pontos.
- Sumário: Breve descrição ao funcionamento e uso do produto, com um limite de 144 caracteres.
- Autor(es): Nome do(s) desenvolvedor(es) ou órgão desenvolvedor do produto.
- Versão: Em qual versão de desenvolvimento o produto se encontra, versões alpha ou beta de um produto devem conter junto ao número da versão um ”a”ou um ”b”respectivamente para ser possível tal indicação.
- Arquitetura: A qual arquitetura o produto está destinado a funcionar, podendo ser mais de uma ou até mesmo todas. Este elemento que irá compor o nome do pacote em si, portanto é requerida a legibilidade de mesmo, bons exemplos são: (Linux x86, Windows 64,...)
- Data de Lançamento: A data em que o pacote será distribuído, em formato DD/MM/AAAA.
- Data de instalação: Referente a data em que o produto foi instalado, também em formato DD/MM/AAAA. Esta data será gerada automaticamente no processo de instalação.
- Tamanho: O tamanho em Bytes do produto, gerado automaticamente no processo de empacotamento.
- Licensa/Assinatura: Identificação da licensa.
- Grupo/Tipo: A qual grupo ou tipo de software o produto pertence. Especifica sua principal característica ou objetivo, auxiliando na sua identificação.
- Fonte: Referência ao código fonte do produto.
- Descrição: Documentação explicando o funcionamento, o propósito e todo outro aspectos importante ao produto em si.

*Além de opcionalmente termos:

- Dependências: Lista das dependências do produto.
- Conflitos: Lista de programas ou extensões que entram em conflito com o produto.

Quanto aos aspectos opcionais do descritor, caso algum ou todos não forem requeridos para especificação, o campo que os indica deverá ser deixado em branco. Além disso é necessário frisar que é requerido que a descrição seja o último campo do descritor para facilitar a escrita e leitura do mesmo.

A fim de alcançar uma melhor acessibilidade e compatibilidade, o formato do descritor será o .txt, sendo assim possível para usuários de qualquer arquitetura acessar o descritor, tanto como desenvolver o mesmo. O nome, novamente a fim da legibilidade, será “Descriptor.txt”.

Um exemplo de descritor pode ser visto no Código 2.

```
1 name: example_plugin
2 summary: An example plugin
3 author: Lucas Nascimento Oliveira
4 version: 0.0.1
5 supported architectures: Linux
6 date of creation: 23/05/2015
7 size: 3000 Kb
8 license: GPL
9 type: examples
10 source: http://www.nononon.com.br
11 dependency:
12 conflicts:
13 description: This is an exemplary plugin, made for better
      understanding of the functionality of this descriptor file.
```

Código 2: Exemplo de descritor

4.5. Documentação do usuário

A documentação do usuário é um arquivo de ajuda que deverá conter instruções sobre o uso, instalação e manuseio geral do produto. Este arquivo possui um formato livre mas deverá trazer informações como o propósito do produto, suas aplicações, como melhor aproveitar suas funções, motivação de uso, restrições e recomendações ao sistema, como instalar ou desinstalar manualmente, como mudar certos aspectos do produto caso possível. Todas instruções de uso e suas recomendações devem ser especificado de forma clara e inteligíveis ao olhos do usuário. Também é bastante recomendado uma lista de instruções de como se comportar diante de possíveis erros, seja no uso, na instalação, ou em qualquer outro aspecto relacionado ao produto.

O Código 3 exemplifica tal documentação.

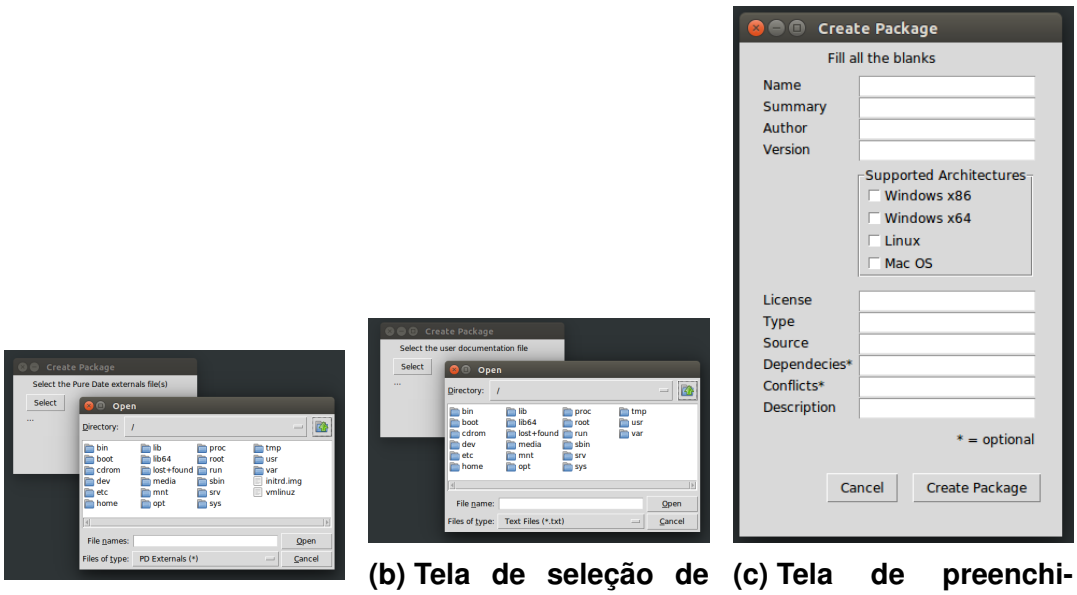
```
1 Installation: There should be at least two ways of instalation, one
      manually and another automatically, these are clear
     instructions to install both ways...
2 HOW-TO: For the proper use of this plugin after installing it...
3 About: This plugin was made for..., it should/must work with...
4 Modding: The user can change these settings of the plugin for these
      purposes...
5 HELP-ME: In case of troubles, the user should take these actions...
```

Código 3: Exemplo de documentação

Juntamente com o descritor e pelos mesmos motivos, o arquivo de ajuda será em formato .txt e chamado ”README”.

5. Resultados obtidos

Além da definição do pacote para plugins do Pure Data, o trabalho desenvolvido até o momento incluiu a criação de duas ferramentas: uma ferramenta para auxiliar o desenvolvedor a criar novos pacotes e uma ferramenta para auxiliar o usuário a instalar novos pacotes.



(a) Tela de seleção **(b) Tela de seleção de documentação** **(c) Tela de preenchimento de metadados**

Figura 2: Ferramenta empacotadora

A ferramenta empacotadora, responsável por criar o pacote, consiste em um plugin TCL/tk que auxilia o desenvolvedor a selecionar os arquivos que compõe o pacote e também a escrever seu descritor. Esta ferramenta cria o arquivo zip com extensão pdpk e define seu nome a partir dos meta-dados informados pelo desenvolvedor, conforme apresentado na Figura 2. Além disso, no processo de empacotamento são adicionadas as informações dos dados não atribuídos ao usuário como o tamanho do arquivo em bytes, além de serem feitas as renomeações necessárias a atender o padrão do pacote. Atendidos os requisitos, o pacote estará pronto para uso, ou em uma melhor colocação, pronto para ser distribuído.

A ferramenta desempacotadora, responsável por extrair os arquivos do pacote e proceder a instalação do mesmo, é um ferramenta semelhante à empacotadora que permite ao usuário selecionar o pacote a ser instalado e a partir do mesmo extrair o(s) arquivo(s) que compõe o plugin no local determinado pelo hospedeiro. O descritor, juntamente com a documentação de usuário, é extraído para uma pasta própria para receber todos os descritores e documentações de todos os pacotes instalados. Nesse processo, ambos arquivos terão seu nome incrementados pelo nome do programa, sua versão e sua arquitetura.

Para melhor representar tal processo veremos que o descritor e documentação antes nomeados assim:

- `Descriptor.txt`
- `README.txt`

Para fins de organização, permitindo uma melhor identificação, após a extração eles serão renomeados para:

- `name-version-supported_architecture-descriptor.txt`
- `name-version-supported_architecture-README.txt`

Com isto, é possível ao usuário verificar os plugins instalados, acessar os metadados dos plugins e gerenciar localmente cada instalação. Este gerenciamento, ainda feito manualmente, será automatizado e incorporado em uma próxima versão da ferramenta e está entre nossos trabalhos futuros.

Quanto ao pacote em si, o mesmo é para uma pasta local chamada repositório local. O repositório local pode ser acessado para qualquer fim, tal como realizar a

reinstalação ou até mesmo o rebaixamento de versão, caso uma nova versão de um programa seja rejeitada pelo usuário que prefere a versão antiga.

Esta ferramenta está sendo desenvolvida em parceria com a lista de desenvolvedores do Pure Data (pd-list-bounces@lists.iem.at) e conta hoje com a colaboração de outros usuários desta plataforma.

6. Conclusão

Este trabalho apresentou os passos iniciais de uma pesquisa cujo foco é o desenvolvimento de um gerenciador de plugins para o ambiente Pure Data. Os passos iniciais foram dados para definir o formato e a estrutura do pacote a ser instalado no ambiente. Para auxiliar a tomada desta decisão, vários formatos de pacotes foram estudados como os pacotes do Debian, Red Hat, NetBeans, Firefox e outros.

O formato de pacote apresentado neste trabalho inclui tomada de decisões de alguns aspectos como: nome do pacote, formato, extensão, descritor e a estrutura interna do pacote. Tais decisões permitiram o desenvolvimento de uma ferramenta para auxiliar o desenvolvedor a empacotar seus plugins e uma ferramenta para auxiliar o usuário a instalar e gerenciar seus plugins. A criação de uma ferramenta para este ambiente propicia uma maneira mais simples de desenvolver projetos no Pure Data de maneira colaborativa, permitindo a desenvolvedores e usuários uma maneira simples de compartilhar códigos e plugins.

Os próximos passos deste projeto são a definição de um repositório remoto, a estrutura de diretório deste repositório e seu descritor além do gerenciador local de pacotes instalados. Uma vez atendida estas definições, será desenvolvida uma ferramenta para carga e descarga de plugins em um repositórios remotos.

Paralelamente ao projeto proposto e indo ao encontro da necessidade de tal iniciativa, foi iniciado um projeto de propósito semelhante já apresentado à comunidade do Pure Data intitulado de deken[1]. Este projeto visa facilitar o acesso do usuário aos externals do Pure Data através do próprio ambiente de desenvolvimento. Dada a semelhança da natureza do deken ao projeto pdpk aqui proposto, é certo que os mesmos serão mesclados em um projeto único aliando o código desenvolvido com os conceitos aqui apresentados. Além disto, tal fusão pode trazer os benefícios do apoio da comunidade de desenvolvimento do Pure Data, a discussão e troca de ideias e a melhoria do ambiente como um todo. Neste ambiente, os repositórios de plugins será gerenciado pela própria comunidade do Pure Data e ficará disponível no site da própria ferramenta.

Apesar de este trabalho ter sido desenvolvido para o ambiente de programação Pure Data, as definições aqui presente podem auxiliar outros desenvolvedores a criar ferramentas de atualização similares para qualquer outra ferramenta.

7. Agradecimentos

Agradecemos a comunidade de desenvolvimento de software livre e aos desenvolvedores anônimos do Pure Data cujo código disponível na Internet permitiram e motivaram este trabalho. Um agradecimento especial ao autor do Pure Data, Miller Puckette, por ter desenvolvido a ferramenta que é objeto deste estudo como FLOSS.

[1] `https://github.com/pure-data/deken`

Referências

Brinkmann, P., Kirn, P., Lawler, R., McCormick, C., Roth, M., and Steiner, H.-C. (2011). Embedding pure data with libpd. In *Proceedings of the Pure Data Convention*, volume 291.

Chrome (2015). What are extensions? `https://developer.chrome.com/extensions`.

Eclipse.org (2003). Eclipse plataform technology overview. `http://www.eclipse.org/whitepapers/eclipse-overview.pdf`.

Eduardo, K. (2013). Começando com as extensões do firefox. `https://developer.mozilla.org/pt-BR/docs/XUL/School_tutorial/Comecando_com_as_Extensoes_do_Firefox`.

Ferreira, R. E. (2006). Gerenciamento de pacotes de software no linux. In *Novatec Editora*, pages 13–15.

MENESES, R. C. and Huzita, E. H. M. (2009). Disen-updater: Um mecanismo de atualização dinâmica de ferramentas para um ambiente de desenvolvimento distribuído de software. In *Simpósio Brasileiro de Engenharia de Software. III Workshop de Desenvolvimento Distribuído de Software*, Fortaleza.

Myatt, A., Leonard, B., and Wielenga, G. (2008). Downloading, installing, and customizing netbeans. *Pro NetBeans^{TM} IDE 6 Rich Client Platform Edition*, pages 1–24.

Novell, I. (2011). Gerenciamento de pacotes. In *Novell, Inc*.

Puckette, M. (2007). *The Theory and Technique of Electronic Music*. World Scientific Publishing Company, Incorporated, Hackensack, N.J.

Puckette, M. et al. (1996). Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41.

Técnicas, A. B. D. N. (1998). *NBR ISO/IEC 12119: Tecnologia de informação - Pacotes de software - teste e requisitos de qualidade*.

Zmölnig, J. (2001). How to write an external for pure-data. *Institute for electronic music and acoustics*.